Tag Syntax

The tags are designed to display dynamic data. To create a input field that displays the property "postalCode", we'd pass the String "postalCode" to the textfield tag.

Creating a dynamic input field

```
<s:textfield name="postalCode"/>
```

If there is a "postalCode" property on the value stack, its value will be set to the input field. When the field is submitted back to the framework, the value of the control will be set back to the "postalCode" property.

Sometimes, we want to pass the dynamic data to a tag. For example, we might want to display a label with the input field, and we might want to obtain the label from the application's messages resources. Accordingly, the framework will parse expressions found in the tag attributes, so that we can merge dynamic data into the tag attributes at runtime. The expression escape sequence is "%{ ... }". Any text embedded in the escape sequence is evalulated as an expression.

Using an expression to set the label

```
<s:textfield key="postalCode.label" name="postalCode"/>
```

The expression language (OGNL) lets us call methods and evaluate properties. The method `getText` is provided by ActionSupport, which is the base class for most Actions. Since the Action is on the stack, we can call any of its methods from an expression, including `getText`.

Non-String Attributes

The HTTP protocol is text-based, but some tags have non-String attribute types, like `bool` or `int`. To make using non-String attributes intuitative, the framework evaulates **all** non-String attributes as an expression. In this case, you do not need to use the escape notation. (But, if you do anyway , the framework will just strip it off.)

Evaluating booleans

```
<s:select key="state.label" name="state" multiple="true"/>
```

Since the attribute `multiple` maps to a boolean property, the framework does not interpret the value as a String. The value is evaluated as an expression and automtically converted to a boolean.

Since it's easy to forget which attributes are String and which are non-String, you can still use the escape notation.

Evaluating booleans (verbose)

```
<s:select key="state.label" name="state" multiple="%{true}"/>
```

Evaluating booleans (with property)

```
<s:select key="state.label" name="state" multiple="allowMultiple"/>
```

Evaluating booleans (verbose with property)

```
<s:select key="state.label" name="state" multiple="%{allowMultiple}"/>
```

`value` is an Object!

Most often, the `value` attribute is set automatically, since `name` attribute usually tells the framework which property to call to set the `value`. But, if there is a reason to set the `value` directly, be advised that **`value` is an Object *NOT* a String**.

⚠️ Since `value` is not a String, whatever is passed to `value` is evaluated as an expression - **NOT** a String literal.

Probably wrong!

```
<s:textfield key="state.label" name="state" value="ca"/>
```

If a `textfield` is passed the value attribute `"ca"`, the framework will look for a property named `getCa`. Generally, this is not what we mean. What we mean to do is pass a literal String. In the expression language, literals are placed within quotes

Passing a literal value the right way

```
<s:textfield key="state.label" name="state" value="%{'ca'}" />
```

Another approach would be to use the idiom `value="'ca'"`, but, in this case, using the expression notation is recommended.

Boiled down, the tag attributes are evaluated using three rules.

1. All *String* attribute types are *parsed* for the "%{ ... }" notation.
2. All *non-String* attribute types are **not** parsed, but evaluated directly as an expression
3. The exception to rule #2 is that if the *non-String* attribute uses the escape notion "%{}", the notation is ignored as redundant, and the content evaluated.

Please remember about *altSyntax* option that can change when value is evaluated as an expression - Alt Syntax

Expression Language Notations

`<p>Username: ${user.username}</p>`	A JavaBean object in a standard context in Freemarker, Velocity, or JSTL EL (Not OGNL).
`<s:textfield name="username"/>`	A username property on the Value Stack.

`<s:url id="es" action="Hello">`
`  <s:param name="request_locale">`
`  es`
`  </s:param>`
`</s:url>`
`<s:a href="%{es}">Espanol</s:a>`
	Another way to refer to a property placed on the Value Stack.

`<s:property` `  value="#session.user.username" />`	The username property of the User object in the Session context.

`<s:select`
`  label="FooBar" name="foo"`
`  list="#{'username':'trillian',`
`  'username':'zaphod'}" />`
	A static Map, as in put("username","trillian").

Disallowed property names

The following names of property are disallowed:

- parameters
- application
- session
- struts
- request
- servletRequest
- servletResponse

The below code will not work:

```
<s:iterator value="parameters"/>
```

```
public class MyAction {

    private String[] parameters;

    public String[] getParameters() {
        return parameters;
    }

}
```

Next: JSP